Introducing Fluvio: The Programmable Data Platform

For Continuous Intelligence

Continuous Intelligence Platform

In the last few years, organizations started to adopt **stream processing** architectures to power a new generation of data-driven services that can detect events, predict behaviors, and respond to customer demand in real-time. As these early pilots become production-ready services, organizations gradually expand these stream processing and analytics pipelines to other services.

In the last few years, organizations started to adopt **stream processing architectures** to power a new generation of data-driven services

Moreover, as data volumes double every few years, organizations that extract valuable and relevant business signals in the shortest amount of time gain a significant competitive advantage. We believe organizations that choose data streaming technology will enjoy an impactful, long-lasting competitive advantage.

Yesterday's Monolithic Stream Processing Platforms

Most stream processing frameworks available today - **Kafka**, **Pulsar**, **Flink**, **Spark**, etc. - were born in the **Big Data era** and designed as monolithic platforms that require sizeable specialized staff to deploy, operate and maintain. Some admin operations such as setting up data sharing or rebalancing stream after config update require an IT ticket to be handled by the operation team.

These Java-based stream processing platforms assume a homogenous and monolithic enterprise development environment of Y2000 where one language rules it all. Some have reluctantly added partial support for Python. Other languages such as Node, Go, and Ruby offer a subset of functionality in independent client libraries. However, Java-derived languages remains the only reliable way to customize stream processing. This barrier makes it difficult for many non-Java developer communities to leverage the power of real-time stream processing. **Github expects 100M developers by 2025**; most of them will be new developers and will not be familiar with Java.

Softbank estimates over 1 trillion devices connected on the Internet of things by 2025 driven by wearables, drones, self-driving cars, inter-connected devices, and more. These networks need stream processing for immediate feedback and real-time analytics for mission-critical decisions. Java-based systems demand significant CPU and memory resources, making them unsuitable for extending stream processing to edge devices.

One of the most significant drawbacks of Java-based stream processing frameworks is the Jar wrapper required for distribution. **Jars** were designed at the dawn of the Internet when browsers were rudimentary HTML readers and programs required runtime applets (aka. sandboxes) to operate. These sandboxes have been riddled with security vulnerabilities, and new browsers are gradually deprecating them. Some frameworks resorted to container technologies such as **Docker** to add another layer of isolation and a workaround for dynamic loading. Unfortunately, the container introduces another layer of **security issues** and introduces more latencies and **cold-startup** time.

The lack of adequate tooling available in the market makes the journey to real-time data stream processing challenging, error-prone, and packed with customizations often reserved for organizations with highly skilled architects and a virtually unlimited budget.

Democratizing data-in-motion

Companies are striving to accelerate digital transformation and become agile data-driven organizations. Yet data is a precious asset often locked down in data lakes or specialized data silos and managed by data teams responsible for storage and safekeeping. As a result, data users lack visibility on what data is available to them and must wait on a lengthy approval process to gain access. This segregated approach to data hinders learning, prevents fast-paced innovation, and ultimately slows down the pace of the business. The **Data Mesh** white paper written by Zhamak Dehghani explains how current data paradigms are ill-suited for modern organizations.

Monolithic data lakes and data silos should be divided into data domains and managed by decentralized teams. These data owners treat data as products and manage the data lifecycle end-to-end. They are responsible for data discovery, quality, and the SLA required by data consumers. This level of autonomy is critical for stream processing, where teams are responsible for generating actionable signals in real-time.

Democratized stream processing requires a self-serviced operational model on top of shared infrastructure. In this model, an infrastructure maintains the shared infrastructure, and the data domain team manages the data. The infrastructure team scales on-demand and re-balances the data dynamically across entire organizations. The data domain teams operate their data stream products independently and export interfaces as needed.

A modern streaming platform must have the following attributes to meet these conditions:

Cloud-Native by design

Cloud-native-based infrastructure is a loosely coupled system where each component can run and scale dynamically. As a result, these systems are well suited for dynamic platforms such as public, private clouds. The cloud-native streaming platform offer:

- Horizontal scale - to meet data elasticity requirements.

- Self-healing - to recover from failures without human intervention.

- Declarative management - to reduce the management burden.

- Kubernetes native - to plug-in native in K8 environments.

Small footprint and resource-efficient

The data-at-motion stream processing must handle an order of magnitude higher data than the products storing data-at-rest. Consequently, stream process platforms must be small enough to boot within milliseconds and operate efficiently on any system architecture. Moreover, it must support a variety of deployments from small organizations to large enterprises. The ideal platform has:

- Small memory footing - to save on cloud resources and run on IoT devices.

- Low latency - to meet real-time latency requirements.

- Leverage multi-core CPU architecture - to operate at maximum performance.

- Fully event-driven with async architecture - to support large I/O.

Companies are striving to accelerate digital transformation and become agile data-driven organizations.

Support Data protection and isolation

Shared infrastructures require a new level of security and privacy protection. In today's **zero-trust** environment, data centers, clouds, and edges are considered **insecure by default**. Products for data-in-motion must segregate data streams and isolate users and teams from each other. Fine-grained context-area rules are used to define:

- Roles - to limit user access based on their role in a group and organization.
- Geo-Locations - to restrict access based on geographical location.
- Identities - to recognize and process data based on their identity.

Stream processing engines with access to data must have a robust **sandboxing environment** that can enforce access control and protect data records from impacting each other.

Full-featured data APIs

All companies are becoming technology companies. Since each team is the owner of their data, they must have API and tools to automate the data product. The self-service stream processing platform must offer granular APIs for developers who want to build robust real-time data services and ease-to-use tools for non-technical users. While SQL-based tools may be adequate for querying data-at-rest or a data lake, they offer limited functionality for developers who need full API access for automation. Data owners need the ability to:

- Customize and manage Data Lifecycle.
- Orchestrate long-running data process.
- Assign declarative API for stream processing.

Development APIs must be available in many widely used programming languages such as Node/JavaScript, Python, Go, Ruby, etc.

Support for data governance

Self-service and de-centralization allow teams to become independent data owners. However, to leverage the team's data as a whole, self-service infrastructure must implement federal governance to aggregate and correlate data cross-organization with policy enforcement. Consumers and regulatory agencies have raised expectations for data protection and securities by regulating access to **PII** and imposing consumer data protection policies such as **GDPR**.

The shared platform is well-positioned to implement federal data governess by baking consistent interoperability and policy standards across teams in conjunction with common access control and audit trails.

Fluvio: Programmable Platform for Data-in-motion

At **InfinyOn**, we are building **Fluvio** as a purpose-built stream processing platform for data-in-motion. Although the platform starts with similar standard stream processing functionalities such as consumer and producer as with legacy Java-based stream processing frameworks, **Fluvio's** performance, scalability, deployment flexibility, and programmability allow building the data-in-motion infrastructure of the future.

Powered by Rust

We start with a strong foundation; The **Rust** language powers **Fluvio**. Rust, a modern programming language built for speed, low overhead, cross-platform interoperability, and code safety. AWS, Mozilla, Google, Facebook, Discord, Dropbox, and others use **Rust** to create a new class of high-performance products, such as browsers, chat servers, network proxies, database servers, real-time systems, and more.

Fluvio needs to perform stream processing at a massive scale. By choosing **Rust**, we gained the following benefits:

Performance by Default

Rust compiles to native code for blazing-fast **performance**. Without **Garbage Collector** pauses, **Rust** can process streams with very low consistent **latency**. The language implements a zero-cost **async** framework capable of handling many concurrent I/O streams with minimum CPU usage. Rust developers can write high-level functional code similar to Java and Python without rolling out hand-crafted machine code to get performance. Performance and zero-cost cost abstraction make Rust an ideal language for data-in-motion.

Safety by Default

Rust is **safe by default** programming language, unlike any other language such as C, C++, Go, or other. It performs many safety checks during compile time instead of discovering fault during production. Rust doesn't allow **NULL**, which happens to be one of the worst in the software mistakes. With borrow checker, Rust prevents **buffer overflow** and dangling pointer that malicious hackers could take advantage of. Microsoft stated that memory safety issues cause **70% of CVE in Windows**. The borrow checkers also check for concurrent logic, which is essential for creating scalable stream processing infrastructure.

> Safety and Performance of Rust enable the Fluvio community to ship a robust, high-performance data streaming platform from day one.

At Infinyon, we are building Fluvio as a purpose-built stream processing platform for data-in-motion.

Cloud-Native Control Plane

Fluvio's control plane took its inspiration from **Kubernetes**. **Fluvio** manages its software components using declarative programming with eventual consistency. With declarative management, the user specifies intent, and the platform attempts to fulfill the request by feedback loop monitoring controllers. For example, wait for additional resources, re-balance workloads, perform reconciliation, trigger self-healing from hardware or software failures, and more.

Fluvio objects can be provisioned through various mechanisms: Kubernetes (kubectl) commands, Fluvio CLI, InfinyOn Cloud, or programmatic admin API available in all supported programming languages.

SmartModules: Programmable Stream Processing

At the heart of all stream-processing frameworks is the **Stream Processor**. SP treats a stream as the core unit of work. Most of the legacy SP uses a fixed pipeline, which only implements fixed subsets of data-in-motion needs:

- Ingestion
- Persistence
- Transmission
- Dispatching
- Computation (filters, maps, aggregates, joins, derivates)

Fluvio's SPU (Stream Processing Unit) comes with a revolutionary programmable stream pipeline. Data-in-motion pipelines need more customization than data-at-rest. For example, filtering and cleaning data, idempotent producers, and others. With a Java-based streaming framework, it is challenging to provide a programmable pipeline with performance and security.

Programmability by WebAssembly

SPU implements programmability by integration with **WebAssembly** technology. **WebAssembly** is a portable binary-code format designed to run in a **secure sandbox**. It is proven W3 technology to bring programmability to software such as **Envoy Proxy**, **Cloudflare worker**, **Microsoft flight simulator**, CDN proxies, and more. **SmartModules** bring WebAssembly technology to real-time data streaming offering an unprecedented level of customization.

SPU's programmability capabilities eliminate the need to stitching together multiple clusters, as seen in other data streaming platforms. SmartStream pipeline runs at native speed, decreasing delays, increasing security, and reducing operational complexity.

Secure by Default

SmartModules separates user stream operations from system stream. All user stream operations are executed as a WebAssembly module in a protected sandbox with separate **memory space** for user operations. Since user modules can only access data supplied by SmartStream, it can't access or modify protected information as PII data.

Fast Inline Computation

With processing time measured in low milliseconds, **SmartModules** offer the fastest and most convenient way to manipulate data-in-motion. A Java-based system performing a similar operation in memory-hungry JARS would see delays from 10 to 100-fold higher with garbage collection and out-of-band management. A container-based system performing a similar function has a significantly higher image size and an out-of-band communication channel, increasing startup time and introducing communication overhead.

Support any development language

WebAssembly supports any language with bindings to the **LLVM toolchain** - Rust, JavaScript, Python, Ruby, and Go. **Fluvio** offers abstractions, templates, utilities, and tools to make it easy and convenient to build and customize stream processing modules.

We believe WebAssembly technology is the key to building high-performance, customizable data streaming platforms.

Solomon Hykes, the creator of Docker, said the **following**:

"If WASM+WASI existed in 2008, we wouldn't have needed to created Docker. That's how important it is. WebAssembly on the server is the future of computing. A standardized system interface was the missing link. Let's hope WASI is up to the task!"

Conclusion

InfinyOn is on a mission to accelerate the world's transition to the real-time economy.

Programmable Data: Cancelling Data Gravity

Democratizing data is really about overcoming data's **gravity** well. Unless it moves, data sits in silos and accumulates gravity. Silo data are difficult to move due to physical storage limitations and fear of exposing ourselves to security breaches. The key to canceling data gravity is programmability. When we apply programmability to data as it moves between services, we can protect, enrich, track and extract information in real-time. Data-in-motion will gradually become an intelligence layer that connects the organization - people, tools, and services. Programmable data, a simple concept that will change the way we manage data.

Fluvio is Open Source. Join Us

Fluvio is an open-source project, and we are committing to make it accessible for everyone. We are at the beginning of our journey. Join us in building the next-generation platform for data-in-motion. Whether you have feedback, ideas, suggestions, and want to become a contributor, reach out. You can find us on **Github** and in **Discord.**

About InfinyOn

InfinyOn, a real-time data streaming company, has architected a programmable platform for data in motion built on Rust and enables continuous intelligence for connected apps. SmartModules enable enterprises to intelligently program their data pipelines as they flow between producers and consumers for real-time services. With InfinyOn Cloud, enterprises can quickly correlate events, apply business intelligence, and derive value from their data. To learn more, please visit infinyon.com.

InfinyOn